Available online at www.sciencedirect.com

SCIENCE DIRECT®

Theoretical Computer Science 341 (2005) 73–90

Theoretical Computer Science

www.elsevier.com/locate/tcs

Oracles for vertex elimination orderings

J. Sawada

Computing and Information Science, University of Guelph, Guelph, Ont., Canada N1G 2W1

Received 11 December 2003; received in revised form 5 January 2005; accepted 20 March 2005

Communicated by G. Ausiello

Abstract

By maintaining appropriate data structures, we develop constant-time transposition oracles that answer whether or not two adjacent vertices in a simple elimination ordering (SEO) or a semiperfect elimination ordering (semiPEO) can be swapped to produce a new SEO or semiPEO, respectively. Combined with previous results regarding convex geometries and antimatroids, this allows us to list all SEOs of a strongly chordal graph and all semiPEOs of an HHDA-free graph in Gray code order. By applying a new amortized analysis we show that the algorithms run in constant amortized time.

Additionally, we provide a simple framework that can be used to exhaustively list the basic words for other antimatroids.

Keywords: Gray code; Antimatroid; Convex geometry; Simple elimination ordering; Semiperfect elimination ordering; Strongly chordal graph; HHDA-free graph; Weak bipolarizable

1. Introduction

A *Gray code* is an exhaustive listing of a combinatorial object where successive objects differ by a constant amount. The ultimate goal for algorithms that produce such listings is for the running time to be proportional to the number of objects generated. Such algorithms are said to be CAT since they run in constant amortized time. In this paper, we combine

E-mail address: sawada@cis.uoguelph.ca.

0304-3975/$ - see front matter © 2005 Elsevier B.V. All rights reserved.
doi:10.1016/j.tcs.2005.03.051

several previous results and several new results to yield an efficient Gray code algorithm that will list:

1. all simple elimination orderings (SEOs) of a strongly chordal graph and
2. all semiperfect elimination orderings (semiPEOs) of an HHDA-free graph.

The algorithm described in this paper is based on a result concerning antimatroids. The *basic words* of an antimatroid are those that are of maximal length. It is shown in [11] that a Gray code exists for the basic words of any antimatroid. An algorithm that produces the Gray code listing is given in [12]. The key ingredient required by the algorithm is an antimatroid-specific *oracle* that answers correctly whether or not an adjacent pair of elements (vertices) in a basic word (ordering) can be swapped to obtain a new basic word. Linear extensions of a poset are an example of the basic words of an antimatroid and a corresponding constant time oracle is given in [12]. The perfect elimination orderings (PEOs) of a chordal graph also form the basic words of an antimatroid, and a corresponding constant time oracle is given in [2]. In this paper, we focus on the more difficult problem of finding a constant time oracle for the basic words of two other antimatroids: the SEOs of a strongly chordal graph, and the semiPEOs of an HHDA-free graph.

It was proved in [11] that if an oracle for a particular antimatroid can be implemented in $O(1)$ time, then the basic words for the antimatroid can be generated in constant amortized time. In this paper we improve this upper bound permitted by the oracle to $O(i)$, where i is discussed in Section 5. This analysis is crucial in proving that we can generate all SEOs and semiPEOs in constant amortized time. In addition to the analysis, we also present a simple framework based on the presentations of the generic algorithm in [12,2], that can be used to generate the basic words for any antimatroid.

The remainder of this paper is presented as follows. In Section 2 we present graph-related definitions focusing on chordal, strongly chordal, and HHDA-free graphs. In Section 3, we discuss convex geometries and use results from [4,5] to show how they relate to our graphs of interest. Then in Section 4, we discuss antimatroids and show how they relate to convex geometries. In Section 5, we present the generic Gray code algorithm outlining the steps required to apply it to any antimatroid. Also in that section we present some important observations about the algorithm and improve the previous analysis. In Section 6, we apply the general framework to PEOs, SEOs, and semiPEOs. In doing so we present constant time oracles for each ordering. We conclude with a brief summary in Section 7.

2. Graph definitions

We start this section with some general graph definitions. We then focus on three graph classes that are related to convex geometries and antimatroids: chordal graphs, strongly chordal graphs, and HHDA-free graphs.

Let $G = (V, E)$ be an undirected graph with vertex set V, where $|V| = n$, and edge set E. Let $N(v)$ denote the neighborhood of a vertex v and let $N[v]$ denote $N(v) \cup \{v\}$. For $A \subseteq V$, we let $G(A)$ denote the subgraph of G induced by A. A complete induced subgraph is called a *clique* and P_k is used to denote an induced path on k vertices. Given an ordering of vertices $f = v_1, v_2, \ldots, v_n$, we let $G_f(i)$ denote the subgraph induced by $\{v_i, \ldots, v_n\}$.

2.1. Chordal graphs

A graph G is *chordal* if every cycle of length 4 or more contains a *chord*—an edge between two nonconsecutive vertices in the cycle. A vertex v is *simplicial* if $G(N(v))$ is a clique, or alternatively, if it is *not* the center of a P_3. An ordering $f = v_1, \ldots, v_n$ is called a PEO if for each $1 \leqslant i \leqslant n$, the vertex v_i is simplicial in $G_f(i)$.

Theorem 1 (*Fulkerson and Gross [7]*). *A graph is chordal if and only if it admits a PEO.*

Theorem 2 (*Dirac [3]*). *Every nontrivial chordal graph has at least* 2 *simplicial vertices.*

2.2. Strongly chordal graphs

A graph G is *strongly chordal* if it is chordal and every even cycle of length 6 or more contains a chord splitting the cycle into two odd length paths. A vertex v is *simple* if the set $\{N[u] : u \in N(v)\}$ can be linearly ordered by set inclusion. Alternatively, a vertex v is simple if for any two vertices $x, y \in N(v)$ either $N[x] \subseteq N[y]$ or $N[y] \subseteq N[x]$. An ordering $f = v_1, \ldots, v_n$ is called a SEO if for each $1 \leqslant i \leqslant n$, the vertex v_i is simple in $G_f(i)$. Observe that a simple vertex is simplicial and hence an SEO is a PEO.

Theorem 3 (*Farber [5]*). *A graph is strongly chordal if and only if it admits an SEO.*

Theorem 4 (*Farber [5]*). *Every nontrivial strongly chordal graph has at least* 2 *simple vertices.*

2.3. HHDA-free graphs

A graph G is HHDA-free (or *weak bipolarizable*) if it does not contain a house, a hole (a cycle of length at least 5), a domino, or an 'A' as an induced subgraph (see Fig. 1). HHDA-free graphs were introduced in [10] as both a generalization of bipolarizable graphs and as a modular extension of chordal graphs. A vertex v is *semisimplicial* if it is *not* a midpoint of any P_4 in G. An ordering $f = v_1, \ldots, v_n$ is called a semiPEO if for each $1 \leqslant i \leqslant n$, the vertex v_i is semisimplicial in $G_f(i)$. It turns out that the existence of semiPEOs does not characterize HHDA-free graphs (in fact, they characterize P_4-simplicial graphs [8]), however every HHDA-free graph admits a semiPEO.

Theorem 5 (*Dragan et al. [4]*). *Every nontrivial HHDA-free graph has at least* 2 *semisimplicial vertices.*

3. Convex geometries

Following the definitions in [6], an *alignment* on a finite set V is a family $\mathcal{F}$ of subsets of V that is closed under intersection and contains both V and the empty set. Elements of $\mathcal{F}$ are considered to be *convex sets* and the pair $(V, \mathcal{F})$ is called an *aligned space*. The smallest

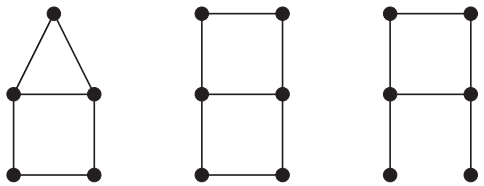

Fig. 1. House, domino, A.

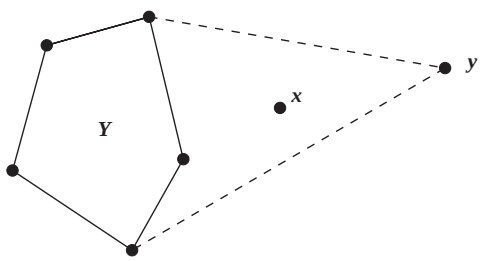

Fig. 2. Illustrating the antiexchange property in $\mathbb{R}^2$.

member of $\mathcal{F}$ containing a given set $S \subseteq V$ is called the *hull* of S. An element y of a set $Y \in \mathcal{F}$ is called an *extreme point* of Y if $Y - \{y\}$ is in $\mathcal{F}$.

A *convex geometry* on a finite set is an aligned space $(V, \mathcal{F})$ such that *Minkowski–Krein–Milman property* is satisfied: every convex set is the hull of its extreme points. Alternatively, a convex geometry is an aligned space that satisfies the *antiexchange property*: for any convex set Y and two distinct points $x, y \notin Y$, if x is in the hull of $Y \cup \{y\}$ then y is not in the hull of $Y \cup \{x\}$. This latter property is an abstraction of a property of convex closures in Euclidean spaces—see Fig. 2.

In this paper, we will focus on four different types of convexity related to graphs: monophonic convexity, geodesic convexity, strong convexity, and m^3-convexity.

3.1. Monophonic convexity

A set of vertices Y is said to be *monophonically convex* (m-convex) if and only if Y contains every vertex on every chordless path between vertices in Y. Note that a vertex v is an extreme point of an m-convex set Y if and only if it is simplicial in $G(Y)$.

Theorem 6 (*Farber and Jamison [6]*). *The monophonic alignment of a graph G is a convex geometry if and only if G is chordal.*

This theorem implies that every m-convex set of a chordal graph is the hull of its simplicial vertices.

3.2. Geodesic convexity

A set of vertices Y is said to be *geodesically convex* (g-convex) if and only if Y contains every vertex on every shortest path between vertices in Y. Again, note that a vertex v is an extreme point of a g-convex set Y if and only if it is simplicial in $G(Y)$. A *gem* is a P_4 with an additional vertex adjacent to all vertices of the P_4.

Theorem 7 (*Farber and Jamison [6]*). *The geodesic alignment of a graph G is a convex geometry if and only if G is chordal and gem free.*

This theorem implies that every g-convex set of a gem-free chordal graph is the hull of its simplicial vertices.

3.3. Strong convexity

A set of vertices Y is said to be *strongly convex* (s-convex) if and only if Y contains every vertex on every even-chorded path whose endpoints are in Y. Note that a vertex v is an extreme point of an s-convex set Y if and only if it is simple in $G(Y)$.

Theorem 8 (*Farber and Jamison [6]*). *The strong alignment of a graph G is a convex geometry if and only if G is strongly chordal.*

This theorem implies that every s-convex set of a strongly chordal graph is the hull of its simple vertices.

3.4. m^3 convexity

A set of vertices Y is said to be m^3-*convex* if and only if for any pair of vertices $x, y \in Y$ each induced path of length at least 3 connecting x and y is completely contained in Y. Note that a vertex v is an extreme point of an m^3-convex set Y if and only if it is semisimplicial in $G(Y)$.

Theorem 9 (*Dragan et al. [4]*). *The m^3-convex alignment of a graph G is a convex geometry if and only if G is HHDA-free.*

This theorem implies that every m^3-convex set of an HHDA-free graph is the hull of its semisimplicial vertices.

4. Antimatroids

A somewhat complementary approach to convex geometries is the shelling of extreme vertices of graphs. The following theorem gives an indication as to how PEOs, SEOs, and semiPEOs relate to convex geometries.

Theorem 10 (*Farber and Jamison [6], Korte et al. [9]*). *If $(V, \mathcal{F})$ is a convex geometry, then $Y \in \mathcal{F}$ if and only if there exists an ordering $v_1, \ldots, v_j$ of $V - Y$ such that v_i is an extreme point of $Y \cup \{x_i, \ldots, x_j\}$ for each $1 \leqslant i \leqslant j$.*

This theorem associates a hereditary language, and in fact an antimatroid, with every convex geometry. To define an antimatroid, we first require some definitions. Given a finite alphabet V, a language $\mathcal{L}$ is a nonempty set of words consisting of letters of V. A language is *simple* if there are no words with repeated letters. The *content* of a word α, denoted $\tilde{\alpha}$, is the set of distinct letters of α. An *antimatroid* is a pair $(V, \mathcal{L})$ such that $\mathcal{L}$ is a nonempty simple language satisfying the following two properties:

1. If $\alpha\beta \in \mathcal{L}$, then $\alpha \in \mathcal{L}$.
2. If $\alpha, \beta \in \mathcal{L}$, where $\tilde{\alpha} \not\subseteq \tilde{\beta}$, then there exists an $a \in \tilde{\alpha}$ such that the $\beta a \in \mathcal{L}$.

To see the direct correspondence between convex geometries and antimatroids we need some extra notation. Given a convex geometry $(V, \mathcal{F})$, let $L(\mathcal{F})$ denote the set of words $\{v_1 v_2 \cdots v_j : V - \{v_1, \ldots, v_i\} \in \mathcal{F}$ for $1 \leqslant i \leqslant j\}$. Given an antimatroid $(V, \mathcal{L})$, let $F(\mathcal{L})$ denote the set system $\{V - \tilde{\alpha} : \alpha \in \mathcal{L}\}$.

Theorem 11 (*Björner and Ziegler [1]*). *If $(V, \mathcal{F})$ is a convex geometry then $(V, L(\mathcal{F}))$ is an antimatroid. Conversely, if $(V, \mathcal{L})$ is an antimatroid then $(V, F(\mathcal{L}))$ is a convex geometry. Furthermore, $L(F(\mathcal{L})) = \mathcal{L}$ and $F(L(\mathcal{F})) = \mathcal{F}$.*

4.1. An example: PEOs

If G is a chordal graph and $\mathcal{F}$ is the family of all m-convex sets in G, then $(V, \mathcal{F})$ is a convex geometry. Now applying Theorems 10 and 11, a word $\alpha = v_1, v_2, \ldots, v_j$ is in $L(\mathcal{F})$ if and only if for each $i = 1, \ldots, j$, the vertex v_i is simplicial in the subgraph induced by $V - \{v_1, \ldots, v_{i-1}\}$. Or more simply, $\alpha \in L(\mathcal{F})$ if it is the prefix of some PEO of G. The *basic words* of an antimatroid are the words of maximal length. Thus, the basic words of the antimatroid $(V, L(\mathcal{F}))$ are precisely the PEOs of G.

Following this same procedure we see that the basic words of an antimatroid corresponding to a strongly chordal graph G are the SEOs of G. Similarly, the basic words of an antimatroid corresponding to an HHDA-free graph G are the semiPEOs of G.

5. A Generic Gray code algorithm

In this section, we will describe a generic Gray code algorithm that can be used to list the basic words of any antimatroid including:

1. all PEOs of a chordal graph [2],
2. all SEOs of a strongly chordal graph, and
3. all semiPEOs of an HHDA-free graph.

In [12], a Gray code algorithm is developed to list all linear extensions of a partially ordered set (which happen to be the basic words of a particular antimatroid). Later, in [11], it is shown that the same algorithm can be used to list the basic words of any antimatroid by simply customizing the initialization step and adding an antimatroid specific oracle. The

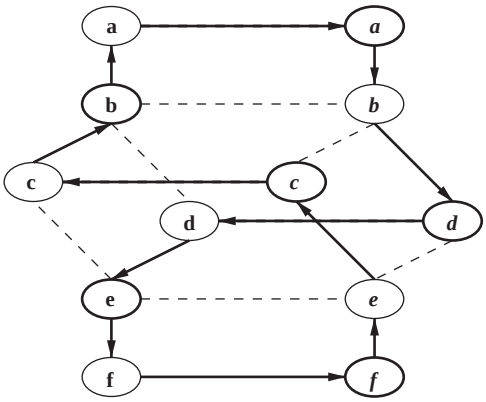

Fig. 3. A Hamilton cycle in the prism of H.

oracle must answer correctly whether or not a specified adjacent pair of elements in a basic word can be swapped to yield another basic word. Since each of the orderings listed at the beginning of this section correspond to the basic words of an antimatroid, we can take advantage of this algorithm. In fact, for the case of PEOs this result has already been applied in [2].

Given an antimatroid, consider the graph $H = (V', E')$ where each vertex in V' corresponds to a basic word and an edge $(u, v) \in E'$ if and only if u and v differ by a single adjacent transposition. The *prism* of H is the graph which results from taking two copies of H and adding edges between the vertices that correspond to the same basic word. For example, if $V' = \{a, b, c, d, e, f\}$ is a set of basic words and $E' = \{(a, b), (b, c), (b, d), (c, e), (d, e), (e, f)\}$, then the prism of $H = (V', E')$ is shown in Fig. 3. In general, H itself may not have a Hamilton cycle, but the prism of H is proved in [11] to always yield a Hamilton cycle. Using this fact, the basic idea behind the generic Gray code algorithm is to trace a specific Hamilton cycle in this graph. Such a traversal visits each basic word exactly twice. However, from Theorem 5.5 in [12], if we print only every second basic word visited in the Hamilton cycle, we obtain each basic word exactly once. By doing this, successive words in the Gray code listing will differ by either 1 or 2 adjacent transpositions.

The basic data structures used by the algorithm are as follows:

- $f[i]$: the ith vertex (element) in the ordering (basic word) $f = v_1, \ldots, v_n$.
- $inv[v]$: the position of the vertex v in the ordering f.
- a_i, b_i: pairs of extreme vertices in the graph induced by $V - \{a_1, b_1, \ldots, a_{i-1}, b_{i-1}\}$.

As initialization, the algorithm must find an initial basic word that is obtained by removing pairs of extreme vertices: a_i, b_i. If n is odd, then the last element in the ordering is the vertex remaining after removing the $\lfloor \frac{n}{2} \rfloor$ pairs of vertices. From Theorems 2, 4, and 5, we know that such vertex pairs exist for antimatroids related to chordal, strongly chordal, and HHDA-free graphs. Additional initialization steps are dependent on the new data structures that are required for the specific antimatroid.

```
procedure Gen ( i: integer );
local j, mrb, mra, mla : integer;  typical: boolean;
begin
    if i = 0 then return;
    Gen( i − 1 );
    mrb := 0;
    typical := false;
    while Swappable( inv[b_i] ) and inv[b_i] ≠ n do begin
        mrb := mrb + 1;
        Move( inv[b_i] );  Gen( i − 1 );
        mra := 0;
        if f[inv[a_i] + 1] ≠ b_i and Swappable( inv[a_i] ) and inv[a_i] ≠ n then begin
            typical := true;
            do
                mra := mra + 1;
                Move( inv[a_i] );  Gen( i − 1 );
            while f[inv[a_i] + 1] ≠ b_i and Swappable( inv[a_i] ) and inv[a_i] ≠ n ;
        end;
        if typical then begin
            Switch( i − 1 );  Gen( i − 1 );
            if Odd( mrb ) then mla := mra − 1;
            else mla := mra + 1;
            for j := 1 to mla do begin
                Move( inv[a_i] − 1 );  Gen( i − 1 );
    end; end; end;
    if typical and Odd( mrb ) then Move( inv[a_i] − 1 );
    else Switch( i − 1 );
    Gen( i − 1 );
    for j := 1 to mrb do begin
        Move( inv[b_i] − 1 );  Gen( i − 1 );
end; end;
```

Fig. 4. Gen(i).

Pseudocode for this Gray code algorithm is given in Fig. 4. The subroutine Move(t) swaps the vertices $f[t]$ and $f[t+1]$ in the ordering. The subroutine Switch(t) swaps the vertices a_t and b_t in the ordering. A call to Switch(t) is only made when a_t and b_t are adjacent. The value a_t always points to the leftmost vertex of the pair, so the values of a_t and b_t are also swapped in this subroutine. These two subroutines are shown in Fig. 5. The function Swappable(t) is the antimatroid specific oracle and the routine Printlt() prints out the current ordering f every *second* time it is called. The initial calling sequence to generate all basic words of a given antimatroid is: Init(); Printlt(); Gen($\lfloor \frac{n}{2} \rfloor$); Switch($\lfloor \frac{n}{2} \rfloor$); Gen($\lfloor \frac{n}{2} \rfloor$);. For more details about the generic Gray code algorithm consult [2] or [12].

```
procedure Move ( t: integer );
begin
    Update( t );
    swap( f[t], f[t + 1] );
    swap( inv[f[t]], inv[f[t + 1]] );
    PrintIt();
end;

procedure Switch ( t: integer );
begin
    if t ≠ 0 then begin
        Update( inv[a_t] );
        swap( f[inv[a_t]], f[inv[b_t]] );
        swap( inv[a_t], inv[b_t] );
        swap( a_t, b_t );
    end;
    PrintIt();
end;
```

Fig. 5. Move(t) and Switch(t).

To apply this generic Gray code to the basic words of a specific antimatroid (e.g. semiPEOs of an HHDA-free graph), we simply add the following three routines:

1. Init(): a routine to initialize f, inv, and the pairs a_i, b_i, as well as new data structures required by the oracle.
2. Swappable(t): an oracle that correctly answers whether or not elements in positions t and $t + 1$ of the current basic word f can be swapped to obtain a new basic word.
3. Update(t): a routine that will update any new oracle-specific data structures upon a swapping of adjacent elements in positions t and $t + 1$.

In Section 6, we apply this algorithm to PEOs, SEOs and semiPEOs. But first, in the following subsection we discuss the running time of the algorithm.

5.1. Analysis

The original analysis of the generic Gray code algorithm in [11] proves that the algorithm runs in constant amortized time given a constant time oracle. In this section, we make some important observations about the algorithm and improve the lower time bound required by the oracle for the generic algorithm to be CAT. These observations are crucial to proving that we can list all SEOs and semiPEOs in constant amortized time. The following result is proved by showing that the number of recursive calls to Gen(i) is proportional to the number of basic words generated:

Theorem 12 (*Pruesse and Ruskey [11]*). *Let* $(V, \mathcal{L})$ *be an antimatroid with an* $O(1)$ *transposition oracle. Then the basic words of* $(V, \mathcal{L})$ *can be generated in constant amortized time such that each word differs from the next by no more than two transpositions.*

If additional data structures are required by the oracle, then to apply this theorem directly they must also be maintained in constant time per transposition or swap. However, note that after two adjacent elements are swapped either by a call to $\mathsf{Move}(t)$ or $\mathsf{Switch}(t)$ during a call to $\mathsf{Gen}(i)$, a recursive call is immediately made to $\mathsf{Gen}(i-1)$. Now an important observation to make is that this single recursive call immediately spawns $i-1$ recursive calls (with parameters $i-2, i-3, \ldots, 0$) via the recursive call at the beginning of the routine (Fig. 4). Thus for each swap, we can perform $O(i)$ operations to update our data structures and amortize the cost over the i recursive calls. Now because the oracle $\mathsf{Swappable}(t)$ is never questioned more than twice before a swap takes place, the cost of the oracle can also be amortized over the i recursive calls. This proves the following theorem:

Theorem 13. *If there exists a transposition oracle for an antimatroid* $(V, \mathcal{L})$ *that answers correctly in* $O(i)$ *time and whose data structures can be updated in* $O(i)$ *time after a swap during a call to* $\mathsf{Gen}(i)$*, then the basic words of* $(V, \mathcal{L})$ *can be generated in constant amortized time such that each word differs from the next by one or two transpositions.*

Before introducing some specific oracles, we make one more observation about the generic Gray code algorithm. Recall that the initial basic word is created by successively removing pairs a_i, b_i of extreme elements. The following observation can be made by focusing on the parameters in the calls made to $\mathsf{Move}(t)$ and $\mathsf{Switch}(t)$ during a call to $\mathsf{Gen}(i)$.

Observation 1. *After two vertices are swapped in the generic Gray code algorithm during a call to* $\mathsf{Gen}(i)$*, any query to the oracle* $\mathsf{Swappable}(t)$ *that is deeper in the computation will be between two adjacent vertices* x *and* y *in the current basic word where* x *precedes* y *and* $x \in \{a_1, b_1, \ldots, a_i, b_i\}$.

We will see in the next section that this observation will allow us to make only partial updates to the oracle specific data structures, since the first vertex involved in all swaps deeper in the recursion will be restricted.

6. The oracles

In this section, we outline efficient oracles for antimatroids related to chordal graphs, strongly chordal graphs, and HHDA-free graphs. In each case, the number of basic words (PEOs, SEOs, semiPEOs) generated is $\Omega(2^n)$, since there are at least 2 *extreme* (simplicial, simple, semisimplicial) vertices in any induced subgraph. Thus, any polynomial amount of precomputation will not affect the overall running time of the generic Gray code algorithm.

```
procedure Swappable (t: integer );
local x, y : integer;
begin
      x := f[t];    y := f[t + 1];
      return(h_x = h_y + 1 or (x, y) ≠ E);
end;


procedure Update (t: integer );
local x, y : integer;
begin
      x := f[t];    y := f[t + 1];
      if (x, y) ∈ E then begin
            h_y := h_y + 1;    h_x = h_x − 1;
end; end;
```

Fig. 6. Swappable(t) and Update(t) for PEOs.

6.1. PEOs of a chordal graph

A constant time oracle for the perfect elimination orderings of a chordal graph $G = (V, E)$ is described in [2]. To obtain the constant time efficiency, for each vertex v_i in the current ordering f, we maintain the value h_i: the size of the neighborhood of v_i with respect to $G_f(i)$. With this information, along with the basic adjacency information, the oracle and the update routines can be implemented in constant time—see Fig. 6. The oracle is based on the following theorem:

Theorem 14 (*Chandran et al. [2]*). *If $f = v_1, \ldots, v_n$ is a PEO of a chordal graph G then f_j is a PEO of G if only if $(v_j, v_{j+1}) \notin E$ or $h_j = h_{j+1} + 1$.*

The initial ordering can be initialized in linear time—again see [2] for details. Using this ordering, the values for h_i can also be initialized in linear time by visiting the neighborhoods of each vertex.

6.2. SEOs of a strongly chordal graph

In this section, we outline an efficient transposition oracle for simple elimination orderings. We start by outlining the basic requirements for f_j to be an SEO, given that f is an SEO.

Lemma 1. *Let $f = v_1, \ldots, v_n$ be an SEO of a strongly chordal graph G. Then f_j is an SEO if and only if v_j is a simple vertex in $G_f(j)$.*

Proof. By definition, the ordering $f_j = u_1, \ldots, u_n$ is an SEO if and only if u_i is simple in $G_{f_j}(i)$ for $1 \leqslant i \leqslant n$. Now observe that $u_i = v_i$ and $G_{f_j}(i) = G_f(i)$ for all i not equal to j

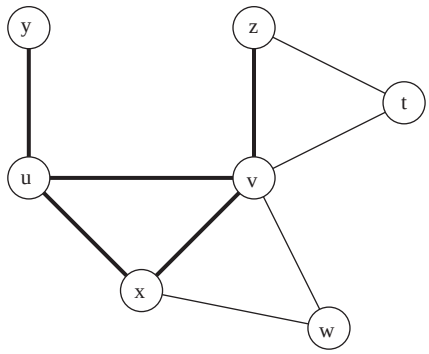

Fig. 7. A forbidden triple (x, y, z) in a strongly chordal graph.

or $j+1$. Thus, since f is an SEO, u_i is simple in $Gf_j(i)$ for all i not equal to j or $j+1$. Also since $v_j = u_{j+1}$ is simple in $G_f(j)$, it will also be simple in the induced subgraph $G_{f_j}(j+1)$. Now because $u_j = v_{j+1}$ and $G_{f_j}(j) = G_f(j)$, the ordering f_j will be an SEO if and only if v_{j+1} is a simple vertex in $G_f(j)$. □

This lemma states that a transposition oracle for SEOs need only test if the vertex v_{j+1} is simple in $G_f(j)$. For the remainder of this discussion, assume that all neighborhoods are with respect to the induced subgraph $G_f(j)$. The task of verifying whether or not v_{j+1} is simple becomes difficult in the case when v_j is adjacent to one or more vertices in $N(v_{j+1})$ but not v_{j+1} itself. In all other cases, v_{j+1} will be simple in $G_f(j)$ as long as it is simplicial—and this can be tested in constant time using the PEO oracle.

Now, assuming that v_{j+1} is simplicial in $G_f(j)$, we consider when v_{j+1} will *not* be simple. From the definition of a simple vertex, this will be the case when there exists two vertices $u, v \in N(v_{j+1})$ such that $N[u] \not\subseteq N[v]$ and $N[v] \not\subseteq N[u]$. In other words, v_{j+1} will *not* be simple in $G_f(j)$ if and only if there exists vertices $u, v, z \in G_f(j)$ such that $u, v \in N(v_{j+1})$ and u is adjacent to v_j, but not z, and v is adjacent to z, but not v_j. Performing such a test will require more than a constant amount of work unless we introduce some new data structures.

The following definition arises from our previous observation. A *forbidden triple* (with respect to strongly chordal graphs) is an ordered triple of unique vertices (x, y, z) that along with two not necessarily unique *joining* vertices u and v form an induced subgraph (a bull) with edge set $\{(x, u), (x, v), (u, v), (u, y), (v, z)\}$. Note that if (x, y, z) is a forbidden triple, then so is (x, z, y). For example, the graph in Fig. 7 illustrates a forbidden triple (x, y, z). In addition, this graph contains the forbidden triples (x, z, y), (x, y, t) and (x, t, y). From the definitions of a forbidden triple and a simple vertex we obtain the following lemma.

Lemma 2. *Let x be a simplicial vertex in a strongly chordal graph G. Then the vertex x is simple if and only if there is no forbidden triple of the form (x, y, z) for any vertices $y, z \in G$.*

Next, we apply the notion of a forbidden triple to SEOs.

Lemma 3. *Let f be a PEO of a strongly chordal graph G. Then f is an SEO if and only if for every forbidden triple $(v_i, v_j, v_k) \in G$ we have $i > min(j, k)$.*

Proof. ($\Rightarrow$) Since f is an SEO, v_i is a simple vertex in $G_f(i)$ and by applying the previous lemma, there is no forbidden triple in $G_f(i)$. Now suppose that there exists a forbidden triple $(v_i, v_j, v_k) \in G$ with joining vertices u and v. The vertex v_i must come after at least one of the other vertices in f since there is no forbidden triple in $G_f(i)$. However, since f is a PEO, both u and v must come after x in the ordering, otherwise v_i would not be simplicial in $G_f(i)$. Thus we must have $i > min(j, k)$.

($\Leftarrow$) The ordering f is an SEO if for each $1 \leqslant i \leqslant n$, the vertex v_i is simple in $G_f(i)$. The fact that v_i is simple in $G_f(i)$ follows from the previous lemma since v_i is simplicial (f is a PEO) and there is no forbidden triple (v_i, v_j, v_k) in $G_f(i)$. Therefore f is an SEO. □

We now present a theorem that will be the basis for an efficient oracle for SEOs.

Theorem 15. *Let $f = v_1, \ldots, v_n$ be an SEO of a strongly chordal graph G and assume that f_j is a PEO. Then f_j is an SEO if and only if there is no forbidden triple (v_{j+1}, v_j, v_k) in G such that $k > j$.*

Proof. ($\Rightarrow$) Let $f_j = u_1, \ldots, u_n$ be an SEO. Thus $u_j = v_{j+1}, u_{j+1} = v_j$ and $u_k = v_k$. Lemma 3 states that every forbidden triple of the form (u_j, u_{j+1}, u_k) in G must satisfy $j > min(j + 1, k)$. Therefore, since $j < j + 1$, we must have $j > k$.

($\Leftarrow$) From Lemma 1 we need only show that v_{j+1} is simple in $G_f(j)$ in order for f_j to be an SEO. From Lemma 2 we must show that there is no forbidden triple (v_{j+1}, y, z) for $y, z \in G_f(j)$. Since v_{j+1} is simple in $G_f(j + 1)$, there is no forbidden triple (v_{j+1}, y, z) where $y, z \in G_f(j + 1)$ (again by Lemma 2). Now since (v_{j+1}, v_j, v_k) is not a forbidden triple for $k > j$, there also is no forbidden triple (v_{j+1}, v_k, v_j). Therefore f_j is an SEO. □

We can now apply this theorem to develop a constant time transposition oracle for simple elimination orderings. First, we must precompute all forbidden triples so that we can determine whether or not (x, y, z) is a forbidden triple in constant time. Second, we must maintain a counter $numBad(x, y)$ for every ordered pair of vertices (x, y) that stores the number of vertices z such that z comes after x in the current ordering and (x, y, z) is a forbidden triple. By maintaining this latter data structure, two adjacent vertices v_j and v_{j+1} in an SEO f can be swapped to produce a new SEO if and only if $numBad(v_{j+1}, v_j) = 0$ and the resulting ordering is a PEO. Thus, since the oracle for PEOs takes constant time, an oracle for SEOs can also be implemented to run in constant time.

Unfortunately, the constant time oracle has a side effect: after a transposition of adjacent vertices v_j and v_{j+1} happens in a call to $\mathsf{Gen}(i)$, we must update the information in $numBad(v_j, z)$ and $numBad(v_{j+1}, z)$ for each vertex z. If (v_{j+1}, z, v_j) is a forbidden triple then we must increment $numBad(v_{j+1}, z)$ by 1. If (v_j, z, v_{j+1}) is a forbidden triple, then since v_{j+1} now precedes v_j in the ordering we must decrement $numBad(v_j, z)$ by 1. If we

update the values for each vertex z then the time required to perform such an update is $O(n)$ in the worst case. However, from Observation 1, we do not need to perform this update for every possible vertex z. This is because queries to the oracle Swappable(t) deeper in the computation tree will only involve looking at *numBad*$(?, z)$ where z is in the set $\{a_1, b_1, \ldots, a_i, b_i\}$. Thus, we need only maintain the correct values for *numBad*(v_{j+1}, z) and *numBad*(v_j, z) for those vertices $z \in \{a_1, b_1, \ldots, a_i, b_i\}$. Since the values for *numBad*(x, y) are global, one might wonder whether or not the values will be accurate when we return from a recursive call. However, since the ordering is the same at the beginning of a recursive call to Gen(i) as the end [12], the values for *numBad*(x, y) will also be the same. This means that the required updates can be done in $O(i)$ time and hence Theorem 13 immediately gives us the following result.

Theorem 16. *The simple elimination orderings of a strongly chordal graph can be generated in constant amortized time.*

Pseudocode for the oracle and update routines are shown in Fig. 8. The update routine makes use of one additional global array, *pair* [], that is computed during the initialization step. For a specified vertex v, the value *pair*$[v]$ holds the index i of the a_i, b_i pair that v belongs to. If n is odd, then the vertex v that does not belong to a pair is assigned *pair*$[v] = \lceil \frac{n}{2} \rceil$. The value *forb*$[x][y][z]$ is set to TRUE if and only if (x, y, z) is a forbidden triple in G. The counter *numBad*$[x][y]$ holds the value for *numBad*(x, y).

The initialization routine for SEOs must perform the following steps:

1. Find an initial SEO f obtained by removing pairs of simple vertices. Using this SEO initialize the pairs a_i, b_i, inv, and *pair*.
2. Initialize *forb*$[x][y][z]$ for all triples of vertices.
3. Initialize *numBad*$[x][y]$ for all pairs of vertices based on the initial ordering f.

Such initialization will be dominated by step 2, which can be done in $O(n^5)$ time.

6.3. SemiPEOs of an HHDA-free graph

The approach for constructing the transposition oracle of semiPEOs is very similar to the construction of the oracle for SEOs described in the previous subsection. Again, we will introduce the notion of a forbidden triple—but this time it will be defined relative to an HHDA-free graph and semiPEOs. We begin by outlining the basic requirements for f_j to be a semiPEO, given that f is a semiPEO. The proof of the lemma is similar to the proof of Lemma 1.

Lemma 4. *Let $f = v_1 \cdots v_n$ be a semiPEO of an HHDA-free graph G. Then f_j is a semiPEO if and only if v_{j+1} is a semisimplicial vertex in $G_f(j)$.*

Given that f is a semiPEO we know that v_{j+1} is semisimplicial in $G_f(j+1)$. Thus, the only way that v_{j+1} will *not* be semisimplicial in $G_f(j)$ is if it is the midpoint of a P_4 in $G_f(j)$ where one of the endpoints must be v_j. Naïvely, we can test this condition in $O(n^2)$ time by considering all other vertices in the remaining two positions of the P_4. However, our goal is an oracle that takes constant time. To achieve this goal, we need to maintain

```
procedure Swappable (t: integer );
local x, y : integer;
begin
    x := f[t];    y := f[t + 1];
    return( (h_x = h_y + 1 or (x, y) ≠ E) and numBad[y][x] = 0 );
end;

procedure Update (t: integer );
local i, j, u, v, x, y : integer;
begin
    x := f[t];    y := f[t + 1];
    if (x, y) ∈ E then begin
        h_y := h_y + 1;    h_x = h_x − 1;
    end;
    i := MIN(pair[x], pair[y]);
    for j := 1 to i do begin
        u := a[j];    v := b[j];
        if forb[x][u][y] then numBad[x][u] := numBad[x][u] − 1;
        if forb[x][v][y] then numBad[x][v] := numBad[x][v] − 1;

        if forb[y][u][x] then numBad[y][u] := numBad[y][u] + 1;
        if forb[y][v][x] then numBad[y][v] := numBad[x][y] + 1;
end; end;
```

Fig. 8. Swappable(t) and Update(t) for SEOs.

some additional data structures. In particular, we again use the notion of a forbidden triple of vertices. A *forbidden triple* (with respect to an HHDA-free graph) is an ordered triple of vertices (x, y, z) such that x is the midpoint of a P_4 with y and z as the endpoints. Thus, if (x, y, z) is a forbidden triple, then so is (x, z, y). Applying this definition to the definition of a semiPEO we obtain the following lemma.

Lemma 5. *Let G be an HHDA-free graph and $f = v_1, \ldots, v_n$ be an ordering of its vertices. Then f is a semiPEO if and only if for every forbidden triple $(v_i, v_j, v_k) \in G$ we have $i > min(j, k)$.*

Proof. ($\Rightarrow$) Suppose that f is a semiPEO and that there exists a forbidden triple $(v_i, v_j, v_k) \in G$ such that $i < min(j, k)$. This implies that there exists a P_4 composed of v_i, v_j, and v_k and some fourth vertex v_l in G where v_j and v_k are the endpoints. This means that (v_l, v_j, v_k) is also a forbidden triple. However, since v_i is a semisimplicial vertex in the graph $G_f(i)$, we must have $l < i$. However, this contradicts the fact that v_l is semisimplicial in $G_f(l)$.

($\Leftarrow$) The ordering f is a semiPEO if for each $1 \leqslant i \leqslant n$, the vertex v_i is semisimplicial in the induced subgraph $G_f(i)$. We are given that for each forbidden triple of the form (v_i, v_j, v_k), $i > min(j, k)$. Thus, each v_i is semisimplicial in $G_f(i)$ since by definition of a forbidden triple it cannot be a midpoint of any P_4 in $G_f(i)$. □

We now present a theorem that will be the basis for an efficient oracle for SEOs.

Theorem 17. *Let $f = v_1, \ldots, v_n$ be a semiPEO of an HHDA-free graph G. Then f_j is a semiPEO if and only if there is no forbidden triple (v_{j+1}, v_j, v_k) in G such that $k > j$.*

Proof. ($\Rightarrow$) Assume that $f_j = u_1, \ldots, u_n$ is a semiPEO. Thus $u_j = v_{j+1}$, $u_{j+1} = v_j$ and $u_k = v_k$. Since v_j comes immediately before v_{j+1} in f_j, the previous lemma implies that for any $k > j$ that (v_{j+1}, v_j, v_k) is not a forbidden triple in G.

($\Leftarrow$) From Lemma 4 we need only show that v_{j+1} is semisimplicial in $G_f(j)$ in order for f_j to be semiPEO. Now since there is no forbidden triple (v_{j+1}, v_j, v_k) such that $k > j$ in f, it follows immediately that v_{j+1} is semisimplicial in $G_f(j)$. □

We can now apply this theorem to develop a constant time transposition oracle for semiPEOs. First, we must precompute all forbidden triples so that we can determine whether or not (x, y, z) is a forbidden triple in constant time. This can be done by searching for all P_4's in $O(n^4)$ time. Second, we must maintain a counter *numBad*(x, y) for every ordered pair of vertices (x, y) that stores the number of vertices z such that z comes after x in the current ordering and (x, y, z) is a forbidden triple. By maintaining this latter data structure, two adjacent vertices v_j and v_{j+1} in a semiPEO f can be swapped to produce a new semiPEO f_j if and only if *numBad*$(v_{j+1}, v_j) = 0$. Thus, by maintaining this additional data structure our oracle responds in constant time by examining the appropriate counter.

As with the SEO case, the constant time oracle has a side effect: after a successful transposition of adjacent vertices v_j and v_{j+1}, we must update the information in *numBad*(v_{j+1}, z) and *numBad*(v_j, z) for each vertex z. It turns out that the updates required are identical as the updates for the SEO case, except we do not have to update the information required to detect simplicial vertices. The analysis is also based on the same reasoning applied to the SEO case.

Theorem 18. *The semiPEOs of an HHDA-free graph can be generated in constant amortized time.*

The oracle and the update routines for semiPEOs are shown in Fig. 9. The initialization steps are virtually the same as the steps required for SEOs, except the notions of forbidden triple are with respect to HHDA-free graphs and semiPEOs.

1. Find an initial SEO f obtained by removing pairs of simple vertices. Using this SEO initialize the pairs a_i, b_i, inv, and *pair*.
2. Initialize *forb*$[x][y][z]$ for all triples of vertices.
3. Initialize *numBad*$[x][y]$ for all pairs of vertices based on the initial ordering f.

Such initialization will be dominated by step 2 which can be done in $O(n^4)$ time.

7. Summary

In this paper we combine several previous results and introduce some new results (the oracles) to obtain efficient (CAT) Gray code listings of:

- SEOs of a strongly chordal graph, and
- semiPEOs of an HHDA-free graph.

```
procedure Swappable (t: integer );
local x, y : integer;
begin
    x := f[t];    y := f[t + 1];
    return ( numBad[y][x] = 0 );
end;

procedure Update (t: integer );
local i, j, u, v, x, y : integer;
begin
    x := f[t];    y := f[t + 1];
    i := MIN(pair[x], pair[y]);
    for j := 1 to i do begin
        u := a[j];    v := b[j];
        if forb[x][u][y] then numBad[x][u] := numBad[x][u] − 1;
        if forb[x][v][y] then numBad[x][v] := numBad[x][v] − 1;

        if forb[y][u][x] then numBad[y][u] := numBad[y][u] + 1;
        if forb[y][v][x] then numBad[y][v] := numBad[x][y] + 1;
end; end;
```

Fig. 9. Swappable(t) and Update(t) for semiPEOs.

Previously, it was known that such efficient listings were possible for PEOs of a chordal graph and for linear extensions of a partially ordered set. An open question is whether listings for the basic words of other antimatroids can also be generated in constant amortized time. One that does not seem trivial is the antimatroid obtained by shelling the vertices on the convex hull of a set of points.

Acknowledgements

Thanks to Frank Ruskey for introducing the problem, to the anonymous referee who pointed out new references, and to NSERC for supporting the research.

References

[1] A. Björner, G.M. Ziegler, Introduction to Greedoids, in: N. White (Ed.), Matroid Applications, Cambridge Univ. Press, Cambridge, 1992.
[2] L.S. Chandran, L. Ibarra, F. Ruskey, J. Sawada, Generating and characterizing the perfect elimination orderings of a chordal graph, Theoret. Comput. Sci. 307 (2) (2003) 303–317.
[3] G.A. Dirac, On rigid circuit graphs, Abh. Math. Sem. Univ. Hamburg 24 (1961) 71–76.
[4] F. Dragan, F. Nicolai, A. Brandstädt, Convexity and HHD-free graphs, SIAM J. Discrete Math. 12 (1) (1999) 119–135.
[5] M. Farber, Characterizations of strongly chordal graphs, Discrete Math. 43 (1983) 173–189.
[6] M. Farber, R.E. Jamison, Convexity in graphs and hypergraphs, SIAM J. Algebraic Discrete Methods 7 (3) (1986) 433–444.

[7] D.R. Fulkerson, O.A. Gross, Incidence matrices and interval graphs, Pacific J. Math. 15 (1965) 835–855.
[8] C.T. Hoàng, B.A. Reed, Some classes of perfectly orderable graphs, J. Graph Theory 13 (4) (1989) 445–463; Academic Press, New York, 1980
[9] B. Korte, L. Lovász, R. Schrader, Greedoids, Springer, Berlin, 1991.
[10] S. Olariu, Weak bipolarizable graphs, Discrete Math. 74 (1989) 159–171.
[11] G. Pruesse, F. Ruskey, Gray codes for antimatroids, Order 10 (1993) 239–252.
[12] G. Pruesse, F. Ruskey, Generating linear extensions fast, SIAM J. Comput. 23 (2) (1994) 373–386.